

ENCRYPTION/DECRYPTION INSTRUCTION SET ENHANCEMENT

Don Van Dyke

Korbin Van Dyke

5

Stephen C. Purcell

FIELD OF THE INVENTION

10 This invention relates to encryption and decryption,  
and in particular to an encryption/decryption method  
under the Data Encryption Standard.

BACKGROUND OF THE INVENTION

15 The Data Encryption Standard (DES) algorithm is a  
block cipher and specifies a cryptographic algorithm that  
encrypts, using a key, a 64-bit block of plaintext to a  
64-bit block of ciphertext. DES is a symmetric algorithm  
-- i.e., the same algorithm and same key are used to  
decrypt the 64-bit block of ciphertext back to a 64-bit  
20 block of plaintext. DES is described in detail in a book  
by BRUCE SCHNEIER, APPLIED CRYPTOGRAPHY (1996),  
incorporated by reference herein.

25 The goal of DES is to encrypt the data such that  
every bit of the ciphertext depends on every bit of the  
data and every bit of the key. DES is intended to  
achieve, after a number of "rounds", zero correlation  
between the ciphertext and the original data or key. DES  
accomplishes this goal using two basic techniques of  
cryptography - confusion and diffusion. At the simplest  
30 level, diffusion is achieved through numerous  
permutations and confusion is achieved through XOR  
operations.

35 In DES, a 56-bit key is derived from a 64-bit key by  
omitting every eighth bit (The omitted bits can be used  
as parity to enhance data integrity). Security in DES  
relies upon the 56-bit key, which can be any 56-bit  
number and can be changed at any time. From this 56-bit  
key, 16 different 48-bit subkeys are created for use in

16 DES rounds. Figure 1 shows a DES algorithm operating on 64-bit of plaintext 110. As shown in Figure 1, the DES algorithm consists of an initial permutation (IP) operation 112 and a final permutation ( $IP^{-1}$ ) operation 120, and 16 rounds of encryption operations 114-1 to 114-16. After IP operation 112, plaintext 110 is divided into 32-bit right portion  $R_0$  and 32-bit left portion  $L_0$ . Thereafter, 16 rounds of an identical operation (including "Function f", explained below) are applied on permuted plaintext 110 using subkeys  $K_1$  through  $K_{16}$  (subkeys are explained in further detail below).  $IP^{-1}$  operation 120 provides ciphertext 122, thereby completing the DES algorithm.

IP operation 112 and  $IP^{-1}$  operation 120 provide no additional security. During IP operation 112, a DES integrated circuit loads a 64-bit datum.  $IP^{-1}$  operation 120 is an inverse operation for IP operation 112. Although IP operation 112 and  $IP^{-1}$  120 can be easily implemented in hardware, these operations cannot be efficiently implemented in software. Hence, due to performance considerations, a software implementation of DES often omits IP operation 112 and  $IP^{-1}$  operation 120. While omitting these operations does not compromise security, this modified DES algorithm deviates from the DES standard.

Figure 2 shows a DES round in further detail. As shown in Figure 2, a 56-bit key 210 is divided into two 28-bit key portions 210a and 210b, which are then stored. Then, depending on which of the sixteen rounds is currently being executed, stored key portions 210a and 210b are circularly shifted left by either one or two bits. Forty-eight (48) bits of shifted key portions 210a and 210b are selected in compression permutation operation 212 as the subkey  $K_i$  for that round. Simultaneously, a 64-bit block of text from either IP operation 112 (first round), or the previous round (i.e., (i-1)th round) is divided into 32-bit left portion  $L_{i-1}$  and 32-bit right portion  $R_{i-1}$ . Using expansion

permutation operation 220, right portion  $R_{i-1}$  is expanded to 48 bits. Like IP operation 112, expansion permutation operation 220 can be implemented readily in hardware but cannot be efficiently implemented in software. The 48-bit output value of expansion permutation operation 220 is then combined with the 48-bit key  $K_i$  using XOR operation 225. The 48-bit result of XOR operation 225 is then processed by 8 substitution box (S-box) operations 222, which results in a 32-bit value 226. 32-bit value 226 is then permuted by a permutation box (P-box) operation 224 to provide a 32-bit value 228. Expansion permutation operation 220, XOR operation 225, S-box substitution operation 222 and P-box permutation 224 together constitute Function  $f$ , which is a building block of the DES algorithm. 32-bit output value 228 of Function  $f$  is then combined with left portion  $L_{i-1}$  using an XOR operation 227. The result of XOR operation 227 is to be used as right portion  $R_i$  in the next round. Right portion  $R_{i-1}$  is provided as left portion  $L_i$  in the next round, using a swap operation indicated by reference numeral 230. At the end of the 16th round, right portion  $R_{15}$  of the 15th round becomes the left 32 bits, and right portions  $R_{16}$  becomes the right 32 bits, for  $IP^{-1}$  operation 120.

As the encryption/decryption process of the DES algorithm of Figure 1 is too computationally demanding for a software implementation on a general purpose microprocessor, the DES algorithm is often implemented by an array of identical special purpose modules outside of the microprocessor. However, several drawbacks are inherent in such an approach. First, partitioning the encryption/decryption tasks between the microprocessor and the special purpose modules is complex, especially since different instruction sets are executed by the microprocessor and the special purpose modules. Second, the total silicon area devoted in the integrated circuit for the special-purpose modules is large and costly.

Third, the sheer number of special purpose modules on the integrated circuit causes decentralization of data flow.

Due to the complexity of the DES algorithm, especially expansion and permutation operations, a  
5 software DES implementation is prohibitively slow.

Therefore, a method for implementing the DES algorithm is needed which (a) does not require special purpose modules, (b) combines all data flow into a unified data path, and (c) executes the DES algorithm  
10 quickly and inexpensively.

#### SUMMARY OF THE INVENTION

In accordance with the present invention, several additional instructions are included in the instruction  
15 set of a general purpose microprocessor to operate in conjunction with hardware included in a data path of the general purpose microprocessor. The additional instructions perform a portion of the DES algorithm, in particular, a portion of a DES round. The state  
20 information used at each step of the encryption portion of the DES algorithm is provided in various general purpose registers of the general purpose microprocessor.

In one embodiment, all sixteen 48-bit subkeys are selected prior to the DES step in the general processor  
25 after a 56-bit DES key is known. In another embodiment, each subkey is selected during the round it is used. In yet another embodiment, each subkey is selected during the round it is used, as part of an additional instruction executed by the general purpose  
30 microprocessor.

Hence, the present invention implements the DES algorithm without the special purpose modules of the prior art. In addition, because hardware is used to implement the part of the DES algorithm which cannot be  
35 efficiently implemented in software, the present invention provides improved performance over a software implementation of the prior art. Furthermore, because the general purpose registers store attributes and

parameters of the added instruction of the present invention, data flow is unified.

The present invention is more fully understood in light of the following detailed description taken  
5 together with the accompanying drawings.

#### BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 depicts a Data Encryption Standard (DES) algorithm implementation.

10 Figure 2 depicts one round of the DES algorithm.

Figure 3 is a block diagram 300 schematically depicting instruction execution in a general purpose microprocessor, including scheduling of additional hardware for performing a part of the DES algorithm, in  
15 accordance with the present invention.

Figure 4 is a schematic of the hardware executing DSTEP.

#### DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

20 In one embodiment, the present invention provides, in a general purpose microprocessor, a DES instruction "DSTEP" which carries out Function f in a small amount of additional hardware in a data path, while storing the states of the DES algorithm, i.e., the  $L_i$ 's,  $R_i$ 's and the  
25 subkeys  $K_i$ 's, in general purpose registers. The remainder of the DES algorithm is carried out by general purpose instructions of the general purpose microprocessor. The present invention eliminates the special purpose modules of the prior art and achieves high performance by  
30 executing a part of the DES instruction in the general purpose hardware (e.g., the general purpose registers for storing attributes and parameters, datapath and control) and performing repetitious tasks in the small amount of additional hardware. Under this approach, the present  
35 invention can achieve a speed improvement by an order of magnitude over software implementations of the DES algorithm in the prior art. Data flow is unified by placing the additional hardware in the data path of the

general purpose processor. Instruction DSTEP is defined in Appendix A.

Figure 3 is a block diagram showing instruction execution in a general purpose processor adapted with additional hardware for execution of the "DSTEP" instruction, in accordance with the present invention. The general purpose processor executes multiple instructions in parallel using pipelining. As shown in Figure 3, an instruction is fetched initially from memory at pipeline stage 31 (labeled "F"). In this embodiment, the general processor supports instructions of various lengths. Thus, pipeline stage 32 is provided to allow alignment of the fetched instruction to the proper byte boundary. Pipeline stage 33 performs target prediction for a branch instruction. Branch prediction is carried out in a special "branch" arithmetic logic unit 307. In addition, pipeline stage 33 is provided also to allow an instruction of a selected group of instructions of the instruction set to be converted to another instruction or instructions for more efficient execution. At pipeline stage 34, the loaded instruction is decoded. At pipeline stage 35, operands for the decoded instructions are read from general purpose register file 330. For example, general purpose registers are allocated at run-time to store the  $L_i$ 's,  $R_i$ 's and the subkeys  $K_i$ 's for each round of the DES algorithm. The DES round input and the DES round output are stored in a byte-interleaved form.

The general purpose processor has three arithmetic logic units ("ALUs", shown in Figure 3 as ALUs 302, 304 and 306) with overlapping execution in pipeline stages 36, 37 and 38, which are provided for address generation, memory access and instruction execution stages, respectively. Arrows 311, 321 and 325 represent 192 bits, 256 bits and 256 bits of input data fetched from register file 330 or the bypass mechanism into ALU 302, ALU 304 and ALU 306, respectively. ALU 302 includes an adder 308, a 2-input arithmetic logic unit 310 and a shifter 314. Adder 308 and ALU 310 operate

independently. Adder 308 and ALU 310 are synchronized to address generation pipeline stage 36, which generates addresses for memory access (e.g., fetching an operand from memory). Within the timing of address generation pipeline stage 36, is provided a logic circuit 309, which is adapted for executing the DSTEP instruction. Logic circuit 309 performs Function f and XOR operation 227 described above and is shown in detail in Figure 4. Logic circuit 309 operates in parallel with ALU 310.

Referring to Figure 4, first operand Src1 and second operand Src2 represent two 64-bit registers obtained directly from the register file or on the fly from the bypass structure. Byte 0 through Byte 7 represent the bytes in first operand Src1 and second operand Src2. For example, Byte 0 contains bits 0 through 7 of first operand Src1 and second operand Src2; Byte 1 contains bits 8 through 15 of first operand Src1 and second operand Src2; Byte 2 contains bits 16 through 23 of first operand Src1 and second operand Src2; Byte 3 contains bits 24 through 31 of first operand Src1 and second operand Src2; and so on. Each byte has identical associated circuitry and is cascaded to the circuitry before and after it. For example, circuitry associated with Byte 3 is cascaded to the circuitry associated with Byte 2 and the circuitry associated with Byte 4. Therefore, only the circuitry associated with one byte, i.e., Byte 3, is explained in detail below.

First operand Src1 is the combined left portion  $L_i$  and right portion  $R_i$  which are interleaved. For example, first operand Src1 contains right portion  $R_0$ , left portion  $L_0$ , through right portion  $R_3$  and left portion  $L_3$ . It is noted that right portion  $R_i$  goes through an expansion permutation 220 (Figure 2) which expands the right portion  $R_i$  from 32 to 48 bits. Two bits of second operand Src2 are discarded to perform the compression permutation 212 (Figure 2) which compresses each byte to six bits (e.g., bits 24 through 29), which are part of a subkey  $K_i$ . The set of six XOR operations, representing XOR operation

225 in Figure 2, XOR the portion of subkey  $K_i$  generated above and the expanded right portions  $R_0$  through  $R_3$ . The result of XOR operation 225 is then processed by a 64x4 ROM which represents S-box substitution 222 in Figure 2.

5 The result of S-box substitution 222 is then processed by P-box permutation 224 which is represented by 32 wires on the bottom of the expanded view in Figure 4. The result 228 of P-box permutation 224 is XORed with left portion  $L_0$  through  $L_3$  by XOR operation 227 which is represented by 10 four XOR operations. The result 229 of XOR operation 227 becomes the right portion  $R_1$  through  $R_4$  for the next round and the right portion  $R_0$  through  $R_3$  become the left portion  $L_1$  through  $L_4$  for the next round. The new right portions and left portions are stored in a 64-bit 15 destination register Dest. The combined Byte 3 of first operand Src1 and second operand Src2 is stored as Byte 3, i.e., bits 24 through 31, of destination register Dest.

Referring back to Figure 3, memory access pipeline stage 37 generates memory access requests using the 20 address generated at stage 36 from adder 308. If the memory access can be satisfied from conventional cache 312, the requested data are provided as output values at the end of memory access pipeline stage 37, aligned appropriately via shifter 314. ALU 310 or logic circuit 25 309 provide a second result at the end of the address generation pipeline stage 36, which is piped through memory access pipeline stage 37 to instruction execution pipeline stage 38. The results of ALU 302 are written back into register file 330 according to the timing of 30 instruction execution pipeline stage 38.

Second ALU 304 includes a conventional variable shifter (composed of a shift amount decoder 316 feeding a shift array 318) and 2-input ALU 320. Thus, the execution in shift amount decoder 316 is aligned to 35 memory access pipeline stage 37 and the executions of shift array 318 and 2-input ALU 320 are aligned with instruction execution pipeline stage 38. The results of shift array 318 and 2-input ALU 320 are written back into



register 330 within the timing of instruction execution pipeline stage 38. Shift array 318 and 2-input ALU 320 execute independently.

ALU 306 includes a conventional multiplier 322 and a  
5 conventional 4-input ALU 324. Multiplier 322 has a latency that spans pipeline stages 36 and 37. The output value of multiplier 322 is provided to ALU 324, which provides a 128-bit output value. Thus, execution of multiplier 322 is aligned to both address generation  
10 pipeline 36 and memory access pipeline 37, and execution of ALU 324 and writing back of results into register file 330 are aligned to instruction execution pipeline stage 38.

In one embodiment, instruction DSTEP is executed in  
15 ALU 302. In the particular configuration described above, processing within the first stage of ALU 302 is not necessary but advantageous because the output value is available earlier in the pipeline. Further, the latency of address generation pipeline stage 36 closely  
20 matches the timing of logic circuit 309, so that no modification of timing control of address generation pipeline stage 36 or any other pipeline stage is necessary.

In this embodiment, subkey  $K_i$  is selected using  
25 instructions of the general purpose processor. The programmer can choose to select all 16 subkey  $K_i$ 's when the key value is received, or just before executing the DSTEP instruction. The instructions for key selection can be executed in ALU 302 or 304. Thus, some benefits  
30 of parallel execution can be achieved in some instances, as key selection operations can overlap -- while DSTEP executes in ALU 302, key selection for the next round can execute in ALU 304. Alternatively, a logic circuit for subkey selection can be included in logic circuit 309 to  
35 provide even higher performance. In this embodiment, in the DSTEP instruction, left and right portions  $L_i$  and  $R_i$  and subkey  $K_i$  are passed using three general purpose

[illegible]

5 In this embodiment, bypass mechanisms are provided in ALU 302, so that the results of logic circuit 309 and shifter 314 can each be provided back as input values to ALU 302. If the programmer uses the same corresponding general purpose registers for sources and destinations, all sixteen rounds of DSTEP can be executed using the bypass mechanism -- i.e., no register write back time (i.e., latency of instruction execution pipeline stage 38) is required, thereby providing even higher performance. Bypass mechanisms are also provided elsewhere in ALUs 302, 304, and 306, so results may be immediately used as operands without delaying through instruction execution pipeline stage 38.

-10-